Kapitel 2: Suche

Teil 1

(Dieser Foliensatz basiert auf Material von Mirjam Minor, Humboldt-Universität Berlin, WS 2000/01)

Künstliche Intelligenz und Suche

- Auswahl: Rucksackproblem
- Planung: z.B. Blocksworld, Routen, TSP, Agenten, Roboter
- Problemlöser: Beweise/Ableitungen - Logik/Prolog; VLSI Layout
- intelligente Spielerprogramme - Gewinnstrategien
- Informationsbeschaffung / Recherche - Diagnose

Mögliche Ziele einer Suche:

- finde eine Lösung
- finde alle Lösungen
- finde die kürzeste Lösung
- finde eine optimale Lösung

In allen genannten Fällen kann zudem der Lösungsweg von Interesse sein.

Zustandsraumsuche

Bevor ein Suchverfahren eingesetzt werden kann, muß das Suchproblem zunächst auf adäquate Weise beschrieben werden. Eine Möglichkeit ist die Zustandsraumrepräsentation. Sie enthält:

Zustände – Beschreibungen der verschiedenen möglichen Situationen
Zustandsübergänge – Operatoren die einen Zustand in einen anderen überführen

Es gibt zwei Mengen ausgewählter Zustände: **Startzustände** und **Zielzustände**. Die Zielzustände können implizit (durch Kriterium) oder explizit (durch Aufzählung) gegeben sein.

– die Angabe einer expliziten Zielmenge kann evtl. genauso aufwendig sein wie die Suche selbst

– Interpretation als Graph/Baum

Beispiele für Zustandsraumbeschreibungen

Beweis eines Theorems:
Zustände: Faktenmenge (Lemmata, Zwischenresultate)
Operatoren: Inferenzregeln (Faktenmenge wird erweitert)
Anfangszustand: Axiome
Zielzustände: Faktenmengen, die das Theorem enthalten

Zauber–Würfel:
Zustände: Stellungen
Operatoren: Drehungen
Anfangszustand: Anfangsstellung
Zielzustände: geordnete Stellung

Beispiele für Zustandsraumbeschreibungen

Prolog:
Zustände: Datenbasis + Abarbeitungszustand (offene Subgoals; alternative Klauseln)
Operatoren: Übergang zu einem Subgoal
Anfangszustand: initiales Programm + Anfrage
Zielzustände: Programmziel

Schiebepuzzle:
Zustände: Positionen aller Steine
Operatoren: mögliche Bewegungen des leeren Feldes
Anfangszustand: aktuelle Stellung
Zielzustände: geordnete Stellung

– warum Bewegung des Leerfeldes und nicht der anderen Teile ?

– schwieriger + mehr zu überprüfen

– Modellierung beeinflusst, wie kompliziert die Suche wird

– Feste Abarbeitungsreihenfolge (Inferenz): Tiefe zuerst, Klauselreihenfolge, links-rechts

Generate and Test ?

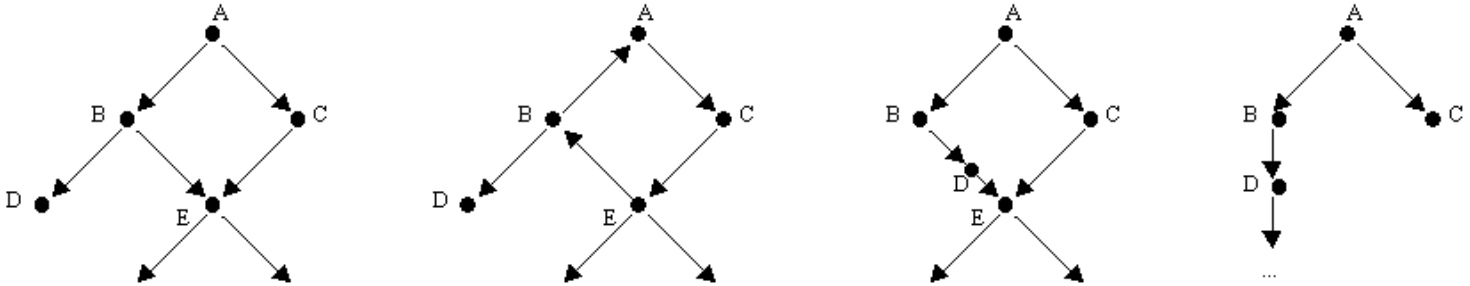
bei den Anfangszuständen beginnend sukzessive Nachfolgezustände generieren und auf Zielbedingung testen.

Problem: kombinatorische Explosion

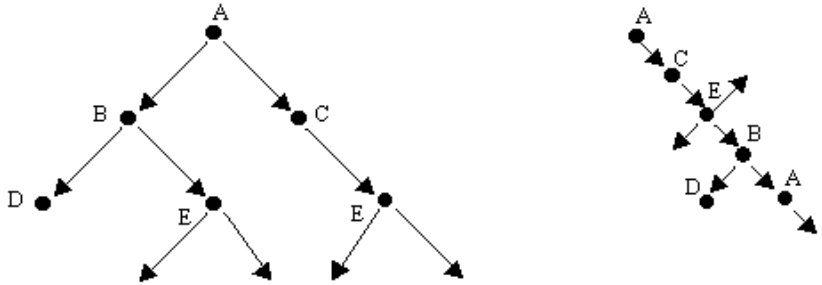
8-er Puzzle: 9! Zustände (davon 9!/2 erreichbar)
bei durchschnittlich 3 Nachfolgern in jedem Zustand und einer typischen Lösungsweglänge von 20 $\Rightarrow 3^{20} \approx$ 3,5 Mrd. Knoten
ungarischer Würfel: rund $4,3 * 10^{19}$ erreichbare Zustände
kürzeste Lösungsfolge max. $40 - 50$ Züge (52 gesichert; 43 vermutet)
Dame: ca. 10^{40} Spiele durchschnittlicher Länge
Schach: ca. 10^{120} Spiele durchschnittlicher Länge
Go: $\leq 3^{361}$ Stellungen

Suchen in Graphen und Bäumen

Probleme: Schlaufen, Zyklen, Abkürzungen, unendliche Pfade



Überführung eines Graphen in einen Baum durch Abwicklung



Schema S zur Suche nach irgendeinem Weg

S0 : Sei z_0 Anfangszustand. Falls dieser bereits Zielzustand ist: EXIT(yes)
OPEN := { z_0 }, CLOSED := { }

S1 : Falls OPEN = { }: EXIT(no)

S2 : Sei z der erste Zustand in OPEN
OPEN := OPEN - {z }, CLOSED := CLOSED + {z }
Bilde Nachfolgermenge SUCC(z), Falls SUCC(z) = { }: GOTO S1

S3 : Falls SUCC(z) einen Zielzustand enthält: EXIT(yes)

S4 : Reduziere SUCC(z) auf NEW(z) durch Streichen nicht weiter zu betrachtender Elemente, Füge New(z) in OPEN ein, GOTO S1

– warum Schema ?

– OPEN enthält generierte, noch nicht expandierte Knoten
– CLOSED enthält die expandierten Knoten

– S0: Start
– S1: negative Abbruchbedingung
– S2: Expandieren
– S3: positive Abbruchbedingung
– S4: Organisation von OPEN

Strategien zur Expansion des Suchraumes

- Expansionsrichtung: vorwärts, rückwärts, bidirektional
- Verwendung von Zusatzinformationen: blinde / heuristische Suche
- Entfernung der Knoten vom Start beachten
- Verwendung aller / einiger Nachfolger
- Modifikationen bzgl. der Elemente in OPEN und CLOSED

Bewertungskriterien für die Qualität eines Suchverfahrens

- Vollständigkeit: wird eine existierende Lösung immer gefunden?
- Optimalität: wird die beste Lösung gefunden?
- Platzbedarf: wieviele Knoten müssen gleichzeitig gespeichert werden?
- Zeitbedarf: wieviele Knoten müssen expandiert werden?
  (im besten Fall, im schlimmsten Fall, im Durchschnitt)

Blinde Suche – Breitensuche (BFS)

in S4 alle Nachfolger ans Ende von OPEN stellen

Platzbedarf b^d; Zeitbedarf $O(b^d)$; vollständig; Lösung mit min. Anzahl von Zustandsübergängen wird zuerst gefunden

Blinde Suche – Tiefensuche (DFS)

in S4 alle Nachfolger an den Anfang von OPEN stellen

Platzbedarf $b*m = O(m)$; Zeitbedarf $O(b^d)$; nicht optimal, bei unendlichen Pfaden im Suchraum nicht vollständig;

b – Verzweigungsgrad; d – Tiefe der Lösung; m – maximale Tiefe des Suchbaumes

Verwendung einer Kostenfunktion

- Jede Zustandsüberführung verursacht Kosten.
  Verlängert sich der Weg, erhöhen sich die Kosten.
  Die Kosten sind nicht negativ und können nicht beliebig klein werden.

- Jedem Operator werden Kosten $c(n \rightarrow \acute{n}) \geq \epsilon > 0$ zugeordnet.

- Kosten eines Pfades: $g(n_0 n_1 ... n_k) = \sum_{i=0}^{k-1} c(n_i \rightarrow n_{i+1})$

- Kosten zum Erreichen von z: $g(z) = Min\{g(s) | s$ ist Weg von z_0 nach $z\}$

Blinde Suche – Gleiche Kosten Suche (UCS — Uniform Cost Search)

- In S4 alle Nachfolger einfügen und OPEN nach Kosten zum Erreichen der Zustände (Wegkosten) sortieren.
- Ermittelte Kosten $g(n)$ stellen eine obere Schranke für die tatsächlichen Kosten $g^*(n)$ dar: $g^*(n) \leq g(n)$
- Platzbedarf $O(b^d)$; Zeitbedarf $O(b^d)$; vollständig; optimal
- BFS ist ein Spezialfall von USC ist ein Spezialfall von A*

Fragen:

– Welches Problem gibt es, wenn keine untere Schranke existiert?

– Wieso ist Breitensuche ein Spezialfall von Gleiche Kostensuche?

Iterative Tiefensuche (IDS) 1/2

- In S4 nur die Nachfolger am Anfang einfügen wenn maximale Suchtiefe c noch nicht überschritten.
- c sukzessive erhöhen.
- Platzbedarf $b * d$; Zeitbedarf $O(b^d)$; vollständig; optimal
- Zeitverhalten in Relation zur Tiefensuche:
  $1 \leq \frac{Zeitbedarf(IDS)}{Zeitbedarf(DFS)} = \frac{b+1}{b-1} \leq 3$

Iterative Tiefensuche (IDS) 2/2

- Knoten in niedrigen Suchtiefen werden wiederholt erzeugt.
- Hiervon gibt es (bei ungefähr gleichbleibendem Verzweigungsgrad) jedoch relativ wenige.
- IDS ist bevorzugte uninformierte Such-Methode bei großem Suchraum, wenn die Tiefe der Lösung nicht bekannt ist.

Aufwand

– alle vorgestellten blinden Suchverfahren haben Zeitverhalten von $O(b^d)$

– Hopcroft/Ullman 1979 "Introduction to Automata Theory": das gilt für jedes Blinde Suchverhalten

– Verzweigungsgrad b und Tiefe der Lösung d werden empirisch abgeschätzt, sie sind domänenabhängig

Verwendung einer heuristischen Schätzfunktion (Heuristik)

Um ein besseres Zeitverhalten zu erreichen, soll Wissen über das konkrete Suchproblem genutzt werden. Zustände, die nahe am Ziel liegen, werden bei der Expansion bevorzugt.

Ideal: Funktion $h^*(n)$, die die **tatsächlichen Kosten** des kürzesten Weges von einem Zustand n zu einem Zielzustand angibt. Die Ermittlung von h^* ist leider oft zu aufwendig.

$h(n)$ liefert eine **Abschätzung der Kosten** für den kürzesten Pfad zum Ziel.

Die Heuristik ist jeweils problemspezifisch.

Anforderungen an eine Heuristik h

- die Heuristik soll stets nicht-negative Werte liefern: $h(n) \geq 0$
- Falls n Zielzustand ist, soll $h(n) = 0$ gelten.
- Bei tatsächlicher Annäherung an einen Zielzustand wird kleinere Distanz signalisiert.
- h soll mit möglichst geringem Aufwand zu errechnen sein.

Seien h_1 und h_2 Heuristiken. h_2 heißt **besser informiert** als h_1, wenn gilt: $\forall n : h_1(n) \leq h_2(n)$

Beispiele für Heuristiken

- die am schlechtesten informierte Heuristik: $\forall n : h(n) = 0$
- für Routenplanung: Luftlinie
- für Schiebepuzzle: Anzahl der falsch liegenden Steine
- für Schiebepuzzle: Manhattan-Distanz
  (Summe der horizontalen und vertikalen Entfernungen aller Steine von ihren Zielpositionen)

Frage: Welche Informationen über das Suchproblem werden jeweils verwendet ?

Heuristische Suche mit schrittweiser lokaler Verbesserung

Bergsteigen (Hill Climbing with Backtracking BTHC)
– In S4 alle Elemente aus SUCC(z) gemäß ihrer Bewertung durch h ordnen und an den Anfang von OPEN stellen.
– nicht vollständig in unendlichen Räumen

optimistisches Bergsteigen (Strict Hill Climbing SHC)
– In S4 nur das am besten bewertete Element aus SUCC(z) an den Anfang von OPEN stellen.
– nicht vollständig

Strahlensuche (Beam Search - BS)
– In S4 die m am besten bewerteten Elemente aus SUCC(z) gemäß ihrer Bewertung ordnen und an den Anfang von OPEN stellen
– nicht vollständig

- Warum Bergsteiger-Metapher?
  - Suchraum als Landschaft interpretieren, h gibt Höhe an

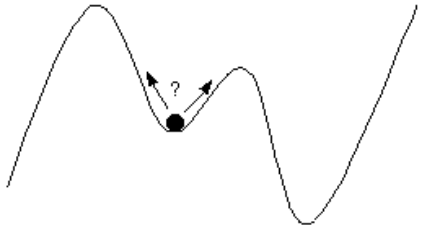
- Warum läuft man nicht nach unten (Abstieg gehört ja auch zu Bergsteigen)?
  - diskrete Version eines Gradienten-Abstiegs (je nachdem ob man die Heuristik als Kosten- oder Qualitätsfunktion ansieht); Optimierung
  - SHC "abwärts" ist analog zu der Pfadfinderregel: Wenn Verlaufen, immer bergab laufen. Irgendwann findet man Tal oder Gewässer (= bevorzugte Siedlungsplätze).

- Raum-/Zeitbedarf sind abhängig von der Heuristik

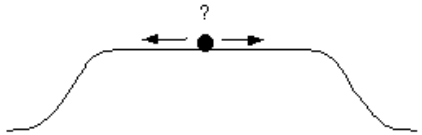
- Übung: Expansion der Suchbäume bei verschiedenen Verfahren zeigen

Probleme beim Bergsteigen

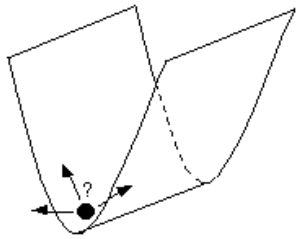
lokale Minima - vorübergehend keine Verbesserung möglich



Plateaus - keine Unterschiede in der Bewertung



enge Talsohlen - vorhandene Operatoren erfassen den Abstieg nicht



– Suche wird langsam bei Abflachung bzw. endet wenn kein Backtracking möglich

Heuristische Suche mit schrittweiser lokaler Verbesserung

Um die genannten Probleme zu umgehen werden die Verfahren modifiziert.

Randomisiertes Verfahren (Random Restart Hill Climbing RRHC)
Wenn kein Fortschritt mehr erzielt werden kann, wird das Verfahren in einem zufällig gewählten Zustand neu gestartet. Der Zustand mit der bislang besten Bewertung wird gespeichert.

Simulated Annealing (Simuliertes Abkühlen)
Mit bestimmter Wahrscheinlichkeit dürfen auch Schritte in Richtungen ausgeführt werden, die nicht zu einer weiteren Minimierung führen. Mit zunehmender Dauer des Verfahrens nimmt diese Wahrscheinlichkeit ab.

Randomisiertes Verfahren

– Globales Minimum wird gefunden, wenn genug Neustarts

– Anzahl der notwendigen Neustarts abh. von Anzahl der lokalen Minima

Simulated Annealing

– Modelliert das Erstarrungsverhalten von Flüssigkeiten

– Erfahrungen vom Stahlkochen: erst stark erhitzen, dann langsam abkühlen ergibt kristalline Struktur, d.h. harten Stahl

– Auswahl: Folgezustand zufällig wählen. Wenn er Abstieg bedeutet, wird er auf alle Fälle gewählt, sonst nur mit vorgegebener Wahrscheinlichkeit.

Weiterer Ansatz: bisherige Richtung zu einem gewissen Teil beibehalten lassen (kontinuierlicher Raum); Schrittweite verändern z.B. bei Lernverfahren

Heuristische Suche - Greedy Search (GS)

Idee: Um schneller zum Ziel zu kommen und lokale Minima zu vermeiden, werden alle Zustände in OPEN zur Bewertung herangezogen.

"Gierig (greedy)", weil ohne Rücksicht auf die Kosten jeweils mit dem vielversprechendsten Zustand weitergemacht wird

In S4 alle Elemente aus SUCC(z) in OPEN übernehmen und die gesamte Liste OPEN gemäß der Bewertung durch h sortieren.

Unvollständig; nicht optimal; Zeit- und Raumbedarf $O(b^m)$; kann durch gute Heuristik deutlich verbessert werden.

Weitere mögliche Eigenschaften einer Heuristik

- Eine Heuristik h heißt **fair**, falls es zu einem beliebigen $k \geq 0$ nur endlich viele Zustände n gibt mit $h(n) \leq k$.

- eine Heuristik h heißt **zulässig** oder **optimistisch**, falls $\forall n : h(n) \leq h^*(n)$.

- eine Heuristik h heißt **konsistent**, falls für alle n gilt: $h(n) - h(n') \leq c(n \rightarrow n')$. Diese Eigenschaft wird auch als **Monotonie-Beschränkung** bezeichnet.

– Fairness + Zyklenvermeidung durch Abgleich mit CLOSE -> Greedy wird vollständig

– bei endlichen Suchräumen ist Fairness automatisch erfüllt

– optimistisch: keine Überschätzung der Kosten

– Konsistenz: geschätzte Distanz zum Ziel schrumpft langsamer, als die Kosten durch diesen Schritt wachsen.

– Heuristiken Anzahl falsche Steine und Manhattandistanz im Schiebepuzzle sind optimistisch

Heuristische Bestensuche - Algorithmus A

A0 : Sei z_0 Anfangszustand. Falls dies bereits Zielzustand ist: EXIT(yes)
OPEN := { z_0 }, CLOSED := { }

A1 : Falls OPEN = { }: EXIT(no)

A2 : Sei z der erste Zustand in OPEN
Falls z Zielzustand ist: EXIT(yes)

A3 : OPEN := OPEN - {z }, CLOSED := CLOSED $\cup$ {z }
Bilde Nachfolgermenge SUCC(z), Falls SUCC(z) = { }: GOTO A1

A4 : für alle $z' \in$ SUCC(z) $g(z')$ gemäß aktuellem Suchbaum neu berechnen
NEW(z) := SUCC(z) - CLOSED
OPEN := OPEN $\cup$ NEW(z), OPEN nach aufsteigendem $g(z')+h(z')$ sortieren,
Wiederholungen in OPEN streichen
GOTO A1

– Änderungen gegenüber dem bisherigen Schema: Schritt 2 und 3 getauscht und modifiziert; Schritt 4 komplexer

– ohne genauere Angaben zur Heuristik lässt sich wenig über das Verhalten von A sagen

Heuristische Bestensuche - Algorithmus A*

In der **weichen Form** von A* wird verlangt, daß die Heuristik **zulässig** ist. A* entsteht aus A durch folgende Modifikation:

A*4 : für alle $z' \in$ SUCC(z) $g(z')$ gemäß aktuellem Suchbaum neu berechnen:
NEW(z) := SUCC(z) - $\{z' \in CLOSED | g(z) \geq g_{alt}(z')\}$
OPEN := OPEN $\cup$ NEW(z), OPEN nach aufsteigendem $g(z') + h(z')$ sortieren, Wiederholungen in OPEN streichen
GOTO A1

Heuristische Bestensuche - Algorithmus A*

Die **harte Form** von A* verlangt eine **konsistente** Heuristik. Dafür kann auf die Tests bereits expandierter Knoten verzichtet werden, denn es gilt: Der A*–Algorithmus in der harten Form untersucht den selben Knoten nicht mehrfach.

A*4 : für alle $z' \in$ SUCC(z) $g(z')$ gemäß aktuellem Suchbaum neu berechnen
NEW(z) := SUCC(z) - CLOSED
OPEN := OPEN $\cup$ NEW(z), OPEN nach aufsteigendem $g(z') + h(z')$ sortieren, Wiederholungen in OPEN streichen
GOTO A1

– A* entspricht A mit zusätzlichen Anforderungen an die Heuristik

– Beweise in Nilsson, 1982: Principles of Artificial Intelligence

– Zeitbedarf ist — ausser bei sehr strengen Anforderungen an h, die in der Praxis meist nicht zu erfüllen sind, immer noch exponentiell in der Länge der Lösung. D.h. dass man optimale Lösungen oft nicht erwarten darf.

– Da A* alle Zwischenergebnisse speichert, ist häufig bereits der Speicher der Engpass.